Methodologies and Tools for Software Vulnerabilities Identification

L. G. C. Barbato[1, 2], and A. Montes[2], N. L. Vijaykumar[1]
[1]Laboratory for Computing and Applied Mathematics - LAC
Brazilian National Institute for Space Research - INPE
C. Postal 515 – 12245-970 – São José dos Campos - SP
BRAZIL

[2]Information Systems Security Division - DSSI
Renato Archer Research Center – CENPRA
13069-901 – Campinas – SP
BRAZIL

E-mail: lgbarbato@lac.inpe.br, antonio.montes@cenpra.gov.br, vijay@lac.inpe.br

Keywords: secure programming, security tests, software security, software vulnerability

INTRODUCTION

The computers connected on Internet are being daily attacked by automatic mechanisms like worms and auto rooters, and by script kiddies that find powerful tools on the network and test them against random systems. Other situation includes people who want to compromise certain institution and attack it. There are many reasons for it, like: espionage, to have fun, to use computer resources, to gain money, etc.

Generally, the objectives of the attacks are the denial of service (DoS), an information manipulation or a non authorized access on the target system. The denial of service is an attack whose purpose is to deny a victim to use a computer resource. The information manipulation attack may be done by hijacking the communication. The non authorized access is achieved through the attainment of authorized people's credentials or through a software vulnerability exploration.

The software vulnerability exploration is the main reason for compromising systems. As said by Viega and McGraw, program vulnerabilities are the most explored. The NVD/NIST data show that between January and April in 2006, at least 64% of the vulnerabilities reported is due to insecure codification. And 9090 database entrances of 16317 are because the programmers do not validate inputs (55.77% of the total). In the OSVDB, 8350 of 13800 are also consequences of the same problem (59.78%).

The bad software codification is the first cause of software vulnerability. Many software houses develop insecure codes, and there are many reasons for that. In general, companies are contracted to specify, to design, to develop and to test software with short deadlines that can cause in many times, terrible errors, for example, the non choice for secure functions to do certain jobs, or the incorrect use of programming language APIs. Besides in the most of cases, security is not part of the project.

The main insecure programming problem is the input validation, as can be seen in the NVD statistics. The program data input can be done in different ways like execution parameters, file and keyboard read, in inter process communication (shared memory, pipes …), network sockets, etc.

The incorrect or non input validation can generate unexpected situations like malicious code execution and buffer overflows. There are many situations in which the software needs to execute operating system commands or to get database data based on the parameters passed to it. Among these parameters there might be some malicious commands to execute improper programs or operations not permitted on databases.

The utilization of secure programming techniques is important to assure the software security. But it alone may not be enough. The analysis process is also a fundamental step to look for security problems. There are two main techniques of analysis to solve this problem: white box and black box. The white box analysis verifies the software source code while the black box tests the running program.

The methodology that is being proposed aims to identify software vulnerabilities without the source code, in other words, this methodology is based on black box analysis. With this, it should be possible to find the best way to discover vulnerabilities based on software characteristics and acceptable risks where the software will run.

THE PROBLEM

Nowadays, software is very important to help businesses of the companies and, in some cases they cannot be accomplished without it. On the other hand, software, in most of the cases, presents serious problems in usability, functionality, availability and security, causing serious financial losses.

The software tendency is to work in architectures where they can be accessed through the Internet. This new software vision permits that people performing their jobs in other places and not necessarily inside the company. With the e-commerce approach, the companies can optimize purchase, sales, and product delivery process.

The Internet is drawing more attention from users because they don't need to go out to do certain personal activities. With the Internet, people can buy products, chat, and manipulate money from online banks through their computers. Internet access expenses are already part of many families' monthly budgets, and this technology adoption is one of the main reasons for the companies' investments in e-commerce.

It's known that each innovation has an associated cost and the software security problems are also growing up with these new technologies. The security problems in computational environment is calling people to cybernetic crimes, because this new approach enable people with malicious intentions get money without any risk to their lives, for example by robbing banks.

These security problems are too simple to be explored that even attackers without technical knowledge can do it. Several tools are being developed and published on the Internet, and some of them are also being sold through Internet underground. The commercial hack tools are so powerful that any person can use because of their friendly graphical interfaces. Attackers don't need to have enough knowledge, but they need just to know how to use the mouse to click.

Software houses are not prepared for security software development and they are not paying much attention to this issue. In general cases, the problem is that companies do not know what it means. They do not know what software security is. They do not know what steps need to be followed to develop in a secure way. At least, they don't know what methodology can be used to this kind of development. These and other doubts have not yet been answered in a way that the answers could have been adopted like a standard.

. Nowadays, there is the ISO/IEC 15408 standard which presents some security requirements that software needs to have for assuring some important aspects that the norm defines. One other good thing is that this norm presents a methodology to classify the software analyzing these requirements.

Now, an individual analysis in the software development base cycle (requirements identifications, software analysis, design, implementation and tests) looking for security aspects that can show how big the problem is. Therefore, the security needs to be in all of these stages. With the objective to help in the resolution of this problem, our work will act on the test step. The motivation to choose this stage is based on the quantity of vulnerabilities which are found for people outside the software houses that these companies' developers haven't found. We think that there is a fault on the security tests which have being done for software companies. All the cycle stages listed are very important to have better security, but if the tests are correctly applied, the probability of attackers in finding vulnerability will decrease.

SOLUTION

The methodology which is being proposed aims to help testing the software vulnerabilities. This methodology will cover from the very initial step where the threats are identified until the final report with the results and action plans. The structure of this methodology is composed by six main activities, such as:

1. Threat analysis
2. Risk analysis
3. Test plan elaboration
4. Tests execution
5. Analysis of results
6. Final report development

1. THREAT ANALYSIS

In this step, the threats are identified and analyzed, in other words, we search for conditions that causes incidents which compromise the assets through vulnerabilities exploration. These conditions are treated in the same way of ISO/IEC 15408:

Threat = Agent X Attack Mechanism X Asset

where, the agents can be anybody or anything which will obtain something with the exploration. The attack mechanisms are the methods used by vulnerabilities exploration, for example, brute force, buffer overflows, denial of service, etc. The assets are the target of protection, i.e., something which can be interesting for the agent. If one of these three factors does not appear, then the threat will not exist. To help the threat analysis, the software characteristics need to be identified. This identification can be done through questionnaire, for example.

Example of interesting characteristics:

- Client/Server architecture
- Web Service
- Distributed System
- Embedded Software
- Allow remote administration
- There is access control
- Etc.

All the relevant characteristics need to be identified to enable the analysis of the real threats and to avoid situations like:

Threat = Attacker trough a SQL injection get non public data on the database.

where, in this case, the system works locally and does have neither network access nor database.

As a matter of fact, this step is targeted for the real software characteristics identification to help on the threat analysis and test plan elaboration.

2. RISK ANALYSIS

The risk analysis in this methodology will organize the elaboration of the test plan, in the tests execution as well as people allocation to conduct the tests.

If the risk can be understood being the probability of the threats explores the vulnerabilities, then the environment situations where the software will run must be analyzed. This concern is important because some hardware, network and operating system configurations can complicate or avoid vulnerability explorations. It does not matter whether application has vulnerability or there is a threat to explore it as long as there are other security measures that can avoid the exploration. For this reason, consider the following formula:

$$Risk = \frac{(Vulnerability \times Threat \times Impact)}{Security\ Measures}$$

In this formula it is possible to observe that whether there are many security measures, the risk tends to be zero. And if there isn't any security measures the denominator is equal to one.

The security measures and impact identification can also be done through questionnaires, for example:

- The operating system has non executable stack
- The kernel is prepared to support some types of attacks
- The software was developed using certain security resources
- The compiler will insert markers in executable code
- There will be dynamic library wrappers to manipulate buffers
- The system will run behind an application firewall
- Etc.

And the impacts, which can be interpreted like the system exploration damages:

- Company image
- Company confidential information's violation
- Clients confidential data access
- Lost of clients
- Company business interruption
- Etc.

So, in this step, the data about the environment and impact are identified to help a test plan elaboration and the results analysis.

3. TEST PLAN ELABORATION

The test planning will be done in this step. In this planning, the information collected in the previous steps are very important to elaborate the test plan. Information such as the software characteristics, the threats, the impacts and security measures will be used to prepare the test cases appropriate for vulnerability detection. The tests cases will be composed of types of tests, such as:

1. Analyze the network input data
2. Analyze the configuration files input data
3. Etc.

Finally, this step will be responsible for the planning of how the tests will be executed in order to analyze the information collected in the previous step.

4. TESTS EXECUTION

After the test plan elaboration with the test cases to be done, the test execution can already be started. This part of the methodology will determine which steps must be followed and which the results will be expected during test execution.

The types of tests, mentioned in the previous step, are followed by procedures which indicate how the tests must be executed. Examples from the previous items:

1. Send the same web formulary data to a server application without the use of client application
   - Modify the data which are being sent and analyzed: the server application behavior, the answers registered in log and the server availability
2. Create a file with different data that the application used to read
   - Change the configuration file with the new one and after that execute the application

The expected results will also be suggested to facilitate the analysis and the development of final report, in such a way that the valuation is more subjective.

Following the previous examples:

1. Send the same web formulary data for a server application without the use of client application. Possible answers:
   - The application doesn't answer
   - The application returns an error code
   - The application allows the operating system command execution
   - The application allows the SQL commands execution

Therefore, in this step the test execution will be done according to the test plan elaborated in the previous step, providing subsidies to next step.

5. ANALYSIS OF RESULTS

In this step, the results of the tests will be analyzed to identify vulnerabilities. The expected results will be associated with the attack mechanisms, which were used in the threat formula, to identify the software vulnerabilities. After that, the risks can be calculated to help the problems solving.

The better approach to calculate the risks haven't chosen yet. This can be qualitative or quantitative, i.e., the formula variables may associate a weight. For example, a SQL injection can have a weight of 2, while denial of service can have a weight of 1. Thus, the risk will be bigger when the software vulnerability involving SQL appears.

Beyond ordering the risks to help identification of priorities, this step can also classify the software per existents risks, helping in this way to choose or purchase a determined product in case this methodology is used for this purpose. The company can also verify whether certain version of a product can or can't be used for the customers.

Therefore, the main objective of this step is the analysis of results obtained by the tests to identify the vulnerabilities and calculate the real risk.

6. FINAL REPORT DEVELOPMENT

In the last step, a standard report will be presented. The data will be distributed in regions of this report where they will be easy to be visualized and analyzed. The better way to elaborate the report will also be studied to facilitate the report generation and interpretation.

As a matter of fact, this step will be targeted to present the analyzed results in an easy way to understand.

FINAL CONSIDERATIONS

In the first stage of the methodology, the attack mechanisms are identified and studied in the technique as well as the relevant software characteristics to correlate these data. For each characteristic we must have a few attack mechanisms.

In the next stage, the security measures which make difficult or disable the explorations are identified and studied to allow an association with the attack mechanisms. Beyond the security measures, the possible impacts are also studied.

Now, a better study about test planning on software engineering lecture is being done in the third stage, because this area is already being well researched for a long time to make the adaptation possible for security tests reality.

The test application procedures are elaborated as well as the test input data in the fourth stage. For each type of test, an execution procedure must be created. The most appropriate data, aiming always the security, are searched and associated for each type of test.

In the fifth stage, analysis, correlation and vulnerability identification methods are created. When necessary, this method will foresee a re-execution of some tests with or without success, or to consider new ones, to improve the conclusions.

The last stage is used to prepare a standard report using the representations techniques researched to present the results data.

REFERENCES

Open Source Vulnerability Data Base (OSVDB), 2006.

Albuquerque, R.; Ribeiro, B. Seguranca no Desenvolvimento de Software, 2002. ISBN 85-352-1095-4.

Arkin, B.; Stender, S.; McGraw, G. Software Penetration Testing. IEEE Security & Privacy, 2005.

Barbato, L. G. C.; Duarte, L. O. Desenvolvimento Seguro de Software. III CTA - Seminário de Tecnologia, Informação e Conhecimento, 2006a.

——. Software Livre e Programação Segura: Verdades e Desafios. III Ciclo de Palestras sobre Software Livre, 2006b.

Barbato, L. G. C.; Duarte, L. O.; Grégio, A. R. A.; Montes, A. Aspectos Práticos da Codificação Segura. GTS 02.2005 - Grupo de Trabalho em Segurança, 2005a.

——. Codificação Segura: Abordagens Práticas. VII SSI - Simpósio de Segurança em Informática, 2005b.

Barbato, L. G. C.; Duarte, L. O.; Montes, A. Programação Segura: Uma Introdução à Auditoria de Códigos. GTS 02.2004 - Grupo de Trabalho em Segurança, 2004.

——. Vulnerabilidades de Software e Formas de Minimizar suas Explorações. GTS 01.2005 - Grupo de Trabalho em Segurança, 2005c.

Barbato, L. G. C.; Montes, A. Segurança de Software: Testes de Caixa Preta. GTS 02.2005 - Grupo de Trabalho em Segurança, 2005.

Blackburn, M.; Busser, R.; Nauman, A.; Chandramouli, R. Model-based Approach to Security Test Automation. Quality Week, 2002.

Cowan, C.; Barringer, M.; Beattie, S.; Kroah-Hartman, G.; Frantzen, M.; Lokier, J. FormatGuard: Atomatic Protection From printf Format String Vulnerabilities. In: Usenix Security Symposium, 10th., 2001, Washington, United States of America. Proceedings... 2001a.

Cowan, C.; Beattie, S.; Wright, C.; Kroah-Hartman, G. RaceGuard: Kernel Protection From Temporary File Race Vulnerabilities. In: USENIX Security Symposium, 10th., 2001, Washington DC, United States of America. Proceedings... 2001b. p. 165 – 172.

Cowan, C.; Pu, C.; Maier, D.; Hinton, H.; Walpole, J.; Bakke, P.; Beattie, S.; Grier, A.; Wagle, P.; Zhang, Q. StackGuard: Adaptive Detection and Prevention of Buffer-Overflow Attacks. In: USENIX Security Symposium, 7th., 1998, San Antonio, Texas, United States of America. Proceedings... 1998. p. 63–78.

Du, W.; Mathur, A. P. Vulnerability Testing of Software System Using Fault Injection, 1998.

——. Testing for Software Vulnerability Using Environment Perturbation, June 2000.

Etoh, H. GCC extension for protecting applications from stack-smashing attacks, 2001.

FIRST. FIRST to host CVSS, 2006

Hoglund, G.; McGraw, G. Exploiting Software: How to Break Code, February 2004. ISBN 0-201-78695-8.

Howard, J. D.; Longstaff, T. A. A Common Language for Computer Security Incidents, October 1998.

Howard, M.; LeBlanc, D.; Viega, J. 19 deadly sins of software security, 2005. ISBN 0-07-2266085-8.

Howard, M.; LeBlanc, D. C. Writing Secure Code, December 2002. ISBN 0-735-61722-8.

Inthurn, C. Qualidade e Teste de Software, 2001. ISBN 85-7502-026-9.x

McGraw, G. Testing for Security During Development: Why we should scrap penetrate-and-patch. IEEE Aerospace and Electronic Systems Magazine, 1998.

MITRE. Common Vulnerabilities and Exposures (CVE), 2006.

National Institute of Standards and Technology (NIST). ICAT, 2005.

One, A. Smashing The Stack For Fun And Profit. Phrack Magazine, v. 49, n. 14, novembro 1996.

Potter, B.; McGraw, G. Software Security Testing. IEEE Security & Privacy, 2004.

Seacord, R. C.; Householder, A. D. A Structured Approach to Classifying Security Vulnerabilities, January 2005.

Shirey, R. RFC 2828: Internet Security Glossary, May 2000.

Stevens, W. R. UNIX network programming Volume 1 Networking APIs: Sockets and XTI, 1998a. ISBN 0-13-490012-X. 9

——. UNIX network programming volume 2 - Interprocess communications, 1998b. ISBN 0-13-081081-9. 9

Symantec. SecurityFocus, 2006.

The Common Criteria Project. ISO/IEC 15408, 1999.

Thompson, H. H.; Whittaker, J. A.; Mottay, F. E. Software Security Vulnerability Testing in Hostile Environments. ACM, 2002.

Vendicator. Stack Shield: A “stack smashing” technique protection tool for Linux, 2000.

Viega, J.; MacGraw, G. Building Secure Software : How to Avoid Security Problems the Right Way. Addison Wesley, 2001. 493 p. ISBN 0-201-72152-X.

Wang, A. J. A. Security Testing in Software Engineering Courses. IEEE, 2004. 21

Weaver, N.; Paxson, V.; Staniford, S.; Cunningham, R. A Taxonomy of Computer Worms, October 2003.